Technical Documentation

Table of Contents

Utility Functions

util/db_util.py

db_open()

- creates a connection to the database that MUST BE CLOSED with db_close()

Input Parameter(s)
None

Return Value
connection

- a connection to the database

db_close(connection)

- closes the connection to the database

Input Parameter(s)
connection

- a connection to the database

Return Value
None

db_query(connection, query)

- queries the database for an SQL query that returns results
- returns such results as a list of lists where each nested list represents a tuple

Input Parameter(s)
connection

- a connection to the database

query

- a string containing an SQL query

Return Value

output

- a list of lists

db_execute(connection, query)

- queries the database for an SQL query that does not return results

Input Parameter(s)

connection

- a connection to the database

query

- a string containing an SQL query

Return Value

None

print_output(output)

- prints the output of a list

Input Parameter(s)

output

- a list

Return Value

None

util/db_helper.py

Tasks Table Functions

get_task(taskID)

- gets a particular tuple in the tasks table in the database based on the task ID and returns a list of lists containing that tuple

Input Parameter(s)

taskID

- an int
- represents the task's ID

Return Value

task

- a tuple containing all information for a given tasks

add_task(taskName,description)

- add a tuple to the tasks table in the database

Input Parameter(s)

taskName

- a string
- represents the title/name of the task

description

- a string
- represents the description of the tasks

Return Value

None

modify_task(taskID, taskName, description, state, employeeID, timeCreated, timeCompleted)

- modifies a tuples in the tasks table in the database with given attributes to be changed

Input Parameter(s)

taskID

- in int
- represents the task ID

taskName

- a string
- contains the task name to which the task's task name should be changed

description

- a string
- contains the description to which the task's description should be changed

state

- a string
- contains the state to which the task's state should be changed

employeeID

- an int
- contains the employee's ID to which the task's employeeID should be changed

timeCreated

- a string
- contains the time to which the task's creation time should be changed

timeCompleted

- a string
- contains the time to which the task's completion time should be changed

Return Value

None

remove_task(taskID)

- removes a particular tuple in the tasks table in the database

Input Parameter(s)

taskID

- an int
- represents the task's ID

Return Value

None

get_all_tasks()

- gets all tuples of the tasks table in the database and returns a list of lists containing those tuples

Input Parameter(s)
None

Return Value
tasks

- a list of lists
- contains all tasks
- nested list is a tuple containing all information for a given tasks

get_completed_tasks()

- gets all tuples of the tasks table in the database which have a Complete state and returns a list of lists containing those tuples

Input Parameter(s)
None

Return Value
tasks

- a list of lists
- contains only tasks that are Complete
- nested list is a tuple containing all information for a given tasks

get_noncompleted_tasks()

- gets all tuples in the tasks table in the database which do not have a Complete state and returns a list of lists containing those tuples

Input Parameter(s)
None

Return Value

tasks

- a list of lists
- contains only tasks that are not Complete
- nested list is a tuple containing all information for a given tasks

get_inprogress_tasks()

- gets all tuples of the tasks table in the database which have an In Progress state and returns a list of lists containing those tuples

Input Parameter(s)
None

Return Value
tasks

- a list of lists
- contains only tasks that are In Progress
- nested list is a tuple containing all information for a given tasks

get_incomplete_tasks()

- gets all tuples of the tasks table in the database which have an Incomplete state and returns a list of lists containing those tuples

Input Parameter(s)
None

Return Value
tasks

- a list of lists
- contains only tasks that are Incomplete
- nested list is a tuple containing all information for a given tasks

get_employee_tasks(employeeID)

- gets all tuples of the tasks table in the database which have a In Progress state and have an employee ID matching the input parameter and returns a list of lists containing those tuples

Input Parameter(s)

employeeID

- an int
- represents the employee's ID

Return Value

tasks

- a list of lists
- contains only tasks that are In Progress that have been claimed by the employee with employeeID
- nested list is a tuple containing all information for a given tasks

update_task_state(taskID, new_state, employeeID)

- updates the tuple in the tasks table matching the task ID with a new state and either an employee ID or a time of completion

Input Parameter(s)

taskID

- an int
- represents the task ID

new_state

- a string
- contains the state to which the task's state should be changed

employeeID

- an int
- contains the employee's ID

Return Value

None

randomly_assign_tasks()

- assign unassigned tasks to employees who have not checked out by changing the *employeeID* and *state* columns

Input Parameter(s)
None

Return Value
None

Items Table Functions

get_item_info(RFID)

- gets the name and price of a tuple in the items table in the database based on the RFID and returns a list of lists containing that tuple

Input Parameter(s)
RFID

- an int
- represents the item's RFID

Return Values
info

- a list of lists
- contains the name and price of item with matching RFID
- nested list is a tuple containing the name and price for the item

get_items()

- gets the name, brand, and price of all tuples in the items table in the database and returns a list of lists containing those tuple

Input Parameter(s)
RFID

- an int

- represents the item's RFID

Return Values
info
- a list of lists
- contains the name, brand, and price of all items
- nested list is a tuple containing the name and price for the item

get_search_result(query)

- gets a list of item names, brands, and prices whose item name match the search query

Input Parameter(s)
query
- a string
- a search query entered to find an item

Return Value
info
- a list of lists
- contains the name, brand, and price of item(s) with names containing the query
- nested list is a tuple containing the name, brand, and price for the item(s)

get_location(id)

- returns the location of an item given its RFID

Input Parameter(s)
id
- an int
- an item's RFID

Return Value
info
- a string
- the location of the item
    - returns a location in the store if available
    - returns the Back Room if not in the store but in the back room

    - returns nothing (empty string) otherwise

Employees Table Functions

get_employee(employeeID)

- gets a particular tuple in the employees table in the database based on the employee ID and returns a list of lists containing that tuple

Input Parameter(s)
employeeID

- an int
- represents the employee's ID

Return Value
employee

- a list
- contains the employee's last name at employee[0] and the employee's first name at employee[1]

Hours Table Functions

get_all_employee_hours()

- gets the employee ID, employee name, days that the employee worked, and the number of hours worked each day that the employee worked

Input Parameter(s)
None

Return Value
hours

- a list of lists
- contains the employee ID, the employee name, the day of the worked hours, and the number of hours worked for that day in descending date for all employees

get_id_employee_hours(employeeID)

- gets the days worked and hours worked for each day for a given employee (based on their employee ID)

Input Parameter(s)
employeeID

- an int
- represents the employee's ID

Return Value
hours

- a list of lists
- contains the day of the worked hours, and the number of hours worked for that day in descending date for the given employee

get_id_null_hours(employeeID)

- gets information for a day that the given employee (based on employee ID) has not checked out

Input Parameter(s)
employeeID

- an int
- represents the employee's ID

Return Value
hours

- a list of lists
- contains the employee ID, check in time, and check out time for a day that a given employee has not checked out

get_latest_employee_hours()

- gets the employee ID, employee name, day, and hours information for hours that have been worked for the most recent day

Input Parameter(s)

None

Return Value
hours

- a list of lists
- contains the employee ID, employee name, day, and hours for employees who have worked hours on the latest day

add_checkout(employeeID)

- updates an entry in the database for a given employee where they have not checked out with the current time as the check out time

Input Parameter(s)
employeeID

- an int
- represents the employee's ID

Return Value
None

add_hours(employeeID)

- adds an entry in the database using the given employee ID and the current time as the check in time

Input Parameter(s)
employeeID

- an int
- represents the employee's ID

Return Value
None

Transactions Table Functions

get_all_item_sales()

- gets the dates, item categories, and amount sold in an item category on a given date

Input Parameter(s)
None

Return Value
item_sales

- a list of lists
- contains date, item category, and the amount of sold in the item category on the specific date

get_all_money_sales()

- gets the date and the amount of money made from sales on that day

Input Parameter(s)
None

Return Value
money_sales

- a list of lists
- contains the date and amount of money made from sales that day

Logins Table Functions

get_password(username)

- gets the password for a given username

Input Parameter(s)
username

- a string
- a user's username

Return Value
password

- a string
- the password for a given user's account or an empty string if it does not exist

get_account_type(username)

- gets the account type for a given username

Input Parameter(s)
username

- a string
- a user's username

Return Value
password

- a string
- the account type for a given user's account or an empty string if it does not exist

get_id(username)

- gets the employee ID for a given username that is an employee/manager account

Input Parameter(s)
username

- a string
- a user's username

Return Value
id

- a string
- the ID for a given employee/manager user's account or an empty string if it does not exist or is not the proper account type

create_user(username, password, accountType, ID, first, last)

- creates an account in the logins table and if the account is for a manager/employee, creates an employee in the employees table

Input Parameter(s)

username

- a string
- the username of the new account

password

- a string
- the username of the new account

accountType

- a string
- the account type of the new account

ID

- a string
- the employee ID of the new employee (if applicable)

first

- a string
- the first name of the new employee (if applicable)

last

- a string
- the last name of the new employee (if applicable)

Return Value

None

ShoppingList Table Functions

get_user_shopping_list(user)

- gets the name, price, and RFID of all items in the given user's shopping list

Input Parameter(s)

user

- a string
- the username of the given user

Return Value
list

- a list of lists
- contains the name, price, and RFID of all items in a user's shopping list

def add_slist_item(user, RFID)

- adds a given item to a given user’s shopping list

Input Parameter(s)
user

- a string
- the username of the given user

RFID

- an int
- the RFID of the item to be added

Return Value
None

remove_slist_item(user, RFID)

- deletes a given item from a given user’s shopping list

Input Parameter(s)
user

- a string
- the username of the given user

RFID

- an int
- the RFID of the item to be removed

Return Value
None

Employee/Manager/Customer Application

login/views.py

This is the most relevant file with regards to coding for the Task Manager. The other files are primarily used to coordinate Django implementation. Relevant HTML files are located in login/templates/. The input parameter `request` is an HTML request for the page.

indexView(request)

- highly associated with homepage.html
- displays a basic homepage where a user can navigate to Employee Login or Customer Login

custLogin(request)

- highly associated with login/cust.html
- displays a customer login page to allow customers enter their username and password to log in
- displays a button to navigate users to a customer registration page

custReg(request)

- highly associated with login/custReg.html
- displays a page allowing customers to register a new customer account by entering a username and password

register(request)

- highly associated with login/cust.html
- creates a tuple in the logins table in the database for the newly register account from custReg
- displays the customer login page (like for custLogin) but with an alert shown at the top that an account was created

empReg(request)

- highly associated with login/empReg.html

- displays a page allowing managers to create a new manager or employee account by entering a first name, last name, username, and password

registerE(request)

- highly associated with login/empReg.html
- creates a tuple in the logins table in the database for the newly register account from empReg
- displays the employee/manage account creation page (like for empReg) but with an alert shown at the top that an account was created

custHome(request)

- highly associated with home/cust.html
- displays a dashboard for logged in customers with options to search for an item or to view their shopping list

empLogin(request)

- highly associated with login/emp.html
- displays a customer login page to allow managers/employees enter their username and password to log in

empHome(request)

- highly associated with home/emp.html and home/man.html
- depending on whether the employee is a manager or just an employee, displays a dashboard
- for employees, displays a dashboard with options to view the task board or to view their own hours/check in or out
- for managers, displays a dashboard to view the task manager, to view analytics, to register new employees, to view the task board, or to view their own hours/check in or out

logout(request)

- highly associated with homepage.html
- changes session text to remove the previously logged in user's username
- displays the homepage

hours(request)

- highly associated with login/hours.html
- shows a list of an employee's hours
- has a button for check in/out
    - shows the check out button is shown if the employee has checked in
    - shows the check in button otherwise

checkedIn(request)

- highly associated with login/hours.html
- checks the employee in
- displays the hours pages like in hours

checkedOut(request)

- highly associated with login/hours.html
- checks the employee in
- displays the hours pages like in hours

task_manager/views.py

This is the most relevant file with regards to coding for the Task Manager. The other files are primarily used to coordinate Django implementation. Relevant HTML files are located in task_manager/templates/task_manager/ with names similar to those of the functions. The input parameter `request` is an HTML request for the page.

index(request)

- highly associated with index.html
- obtains all tasks from the database and sends it to index.html to be displayed as a table with a button for each tuple to send the user to a page where the user can modify most attributes of the tuple
- index.html also displays a button to send the user to a page where the user can create a task, which creates an associated tuple in the database

edit(request)

- highly associated with edit.html
- obtains a given task that the user would like to modify and sends it to edit.html to be displayed
- displays text boxes to allow modification of the task name, description, state, employee ID of the employee who claimed the task (if applicable), time of creation, and time of completion
- displays a button to allow submission of the contents in the text boxes, another button to allow the user to delete the task, and a third button to allow the user to return to the index.html page

modify(request)

- highly associated with modify.html
- obtains the task that the user chose to change in edit.html, which is sent to to modify.html to be displayed
- obtains the task updated with the changed that the user wanted enacted, which is also sent to modify.html to be displayed
- displays a button to allow the user to return to the index.html page

create(request)

- highly associated with create.html

- displays the create.html which displays 2 text boxes—one for entering a task name and one for entering a description—and a button to submit the text

created(request)

- associated with index.html
- creates a task with the user entered task name and description, automatically generated task ID and time of creation (which is the current time)
- obtains all tasks including the created task
- displays index.html but with a notice (which can be closed) that the task has been created

delete(request)

- associated with index.html
- deletes the task which user chose to delete in edit.html in the database
- obtains all tasks excluding the deleted task
- displays index.html but with a notice (which can be closed) that the task has been deleted

task_board/views.py

This is the most relevant file with regards to coding for the Task Board. The other files are primarily used to coordinate Django implementation. Relevant HTML files are located in task_board/templates/task_board/ with names similar to those of the functions. The input parameter `request` is an HTML request for the page.

index(request)

- highly associated with index.html
- displays the index.html page with two buttons—one to claim tasks and one to view tasks
- uses session text to determine employee ID

claim_tasks(request)

- highly associated with claim_tasks.html
- obtains all incomplete tasks and displays them in a table in claim_tasks.html
- displays button by each tuple to allow the user to claim the task
- displays a button to allow the user to return to the index.html page
- uses session text to determine employee ID

view_tasks(request)

- highly associated with view_tasks.html
- obtains and displays all tasks claimed by the given employee (the user)
- displays button by each tuple to allow the user to complete the task
- displays a button to allow the user to return to the index.html page
- uses session text to determine employee ID

claim(request)

- highly associated with claim.html
- updates the task claimed by the user, obtains the task, and displays the task in claim.html and notifies the user that the task has been claimed
- displays a button to allow the user to return to the claim_tasks.html page
- uses session text to determine employee ID

complete(request)

- highly associated with complete.html
- updates the task completed by the user, obtains the task, and displays the task in claim.html and notifies the user that the task has been claimed
- displays a button to allow the user to return to the view_tasks.html page
- uses session text to determine employee ID

analytics/views.py

This is the most relevant file with regards to coding for the Task Manager. The other files are primarily used to coordinate Django implementation. Relevant HTML files are located in analytics/templates/analytics/ with names similar to those of the functions. The input parameter `request` is an HTML request for the page.

index(request)

- highly associated with index.html
- displays a page which allows users to view the latest hours for all employees, view all hours for all employees, view sales figures related to item categories, and view sales figures related to amount sold each day
- only displays the page if the user is logged in as a manager

view_latest_hours(request)

- highly associated with view_latest_hours.html
- shows table of the employee hours for the latest day for which employee hours are available
- only displays the page if the user is logged in as a manager

view_hours(request)

- highly associated with view_hours.html
- shows table of all employee hours
- only displays the page if the user is logged in as a manager

view_money_sales(request)

- highly associated with view_money_sales.html
- shows table of the day and the amount of money made from sales
- only displays the page if the user is logged in as a manager

view_item_sales(request)

- highly associated with view_item_sales.html
- shows table of the days, item categories, and amount of items sold in the given item category for the given day
- only displays the page if the user is logged in as a manager

shopping_list/views.py

This is the most relevant file with regards to coding for the Task Manager. The other files are primarily used to coordinate Django implementation. Relevant HTML files are located in shopping_list/templates/shopping_list/. The input parameter `request` is an HTML request for the page.

index(request)

- highly associated with index.html
- displays a page with a form and a button for item search and a table displaying the items in the user's shopping list
- only displays the page if the user is logged in as a customer

search(request)

- highly associated with search.html
- displays a page with search results from a submitted query in addition to a form and a button for item search
- only displays the page if the user is logged in as a customer

add(request)

- highly associated with index.html
- adds an item to a user's shopping list
- displays a page like in shopping_list/index
- only displays the page if the user is logged in as a customer

delete(request)

- highly associated with index.html
- deletes an item from a user's shopping list
- displays a page like in shopping_list/index
- only displays the page if the user is logged in as a customer

Customer Assistance Terminal

item_locator/views.py

price_checker/views.py

Checkout Terminal

check_out/check_out_helper.py

This is a helper program which holds a linked list program with specific methods for the program

check_out/views.py

This is the most relevant file with regards to coding for the Check_out. The other files are primarily used to coordinate Django implementation. Relevant HTML files are located in check_out/templates/check_out/ with names similar to those of the functions. The input parameter `request` is an HTML request for the page.

index(request)

- highly associated with index.html
- displays the index.html page with Add Items, Remove Item, and Finish buttons

finish(request)

- highly associated with finish.html
- displays the finish.html page with Cash and Credit buttons

add_items(request)

- highly associated with add_items.html which is a copy of index.html with added compatibility functionality
- displays the index.html page with Add Items, Remove Item, and Finish buttons

Returns Terminal